Cloud, Containers, and Config Management

•••

Richard Huang

(content ~~stolen~~ based off the slides by <jaw>)

Topics

What's on the menu?

1. Virtual Machines
2. Containers
3. Configuration

Main Ideas

1. How do I make sure everything is **configured properly** and stays that way?
2. How can I make sure the system **performs reliably with minimal downtime** ?
3. How do I **fix things** when everything breaks

Topics

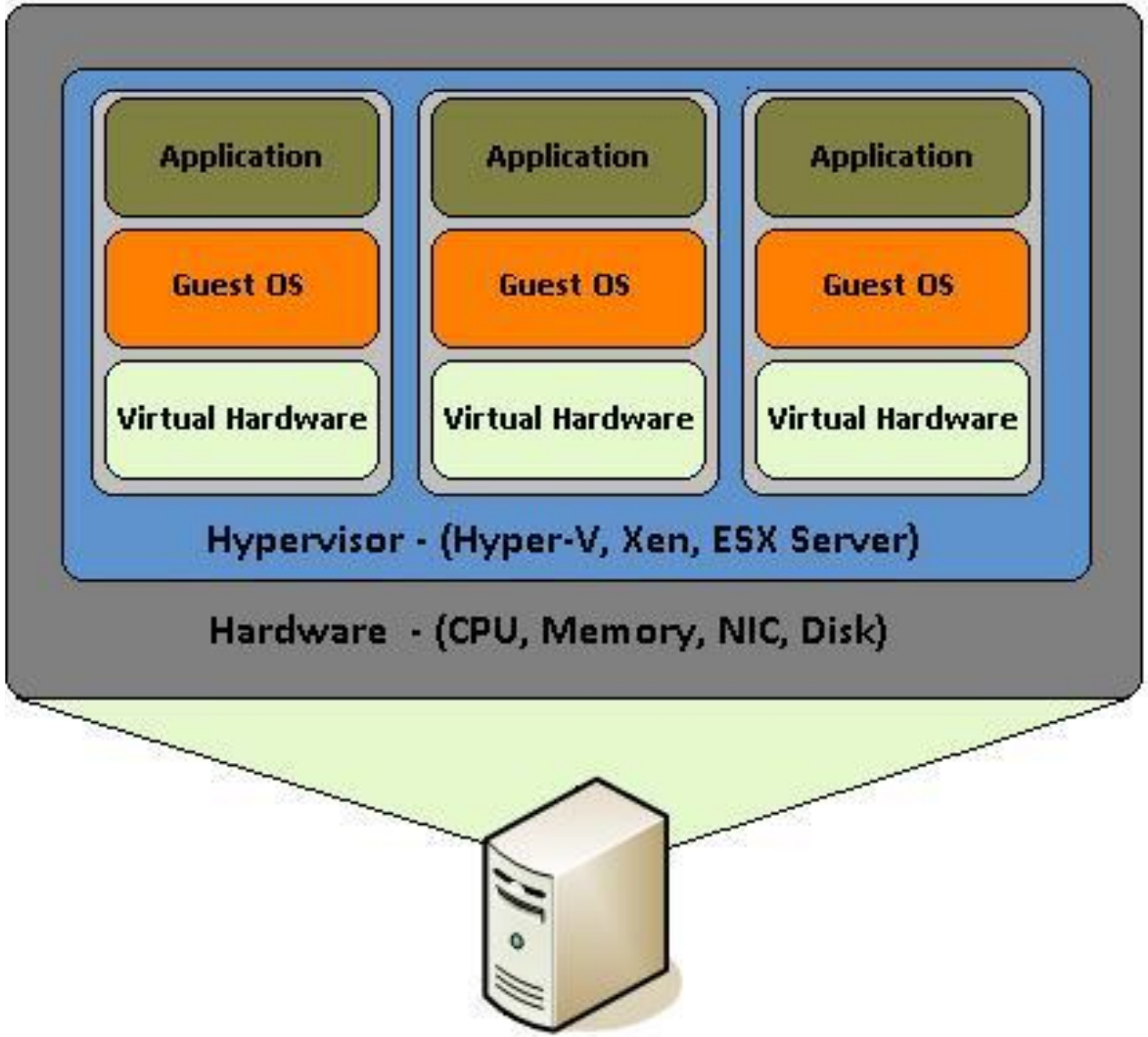
What's on the menu?

1. **Virtual Machines**
2. Containers
3. Configuration

Virtual Machines

A computer inside your computer!

- Abstract away physical hardware via software emulation
- Hypervisor runs multiple VMs
- Isolate applications: better security, stability
- Some overhead: need different guest OS and emulation of virtual hardware for each application
- Takes some time to boot up


Application
Application
Application
Guest OS
Guest OS
Guest OS
Virtual Hardware
Virtual Hardware
Virtual Hardware
Hypervisor - (Hyper-V, Xen, ESX Server)
Hardware - (CPU, Memory, NIC, Disk)

Aside: Layers of Multiplexing/Virtualization

- **Hardware level** : SMT/hyperthreading, multicore computing, SIMD
- **Process-level** : Virtual memory, multithreading
- **Application** : Sandboxed applications
- **Operating System** : Namespaces, kernel-level isolation, BSD Jails, Solaris Zones, Linux Containers
- **Full Virtualization** : Hardware virtualization, Type 1/2 hypervisors (KVM, VMWare, Xen, Hyper-V)

Topics

What's on the menu?

1. Virtual Machines
2. **Containers**
3. Configuration

Containers

- Goal is to provide lightweight isolation by sharing code (libraries) and hardware with host
- Much faster to boot up
- Easy to package applications for consistent deployments
- Common container runtimes: Docker, rkt, LXC
- Pretty sure this is the only class at UC Berkeley that uses Docker

Docker

- Container platform backed by OS-level virtualization
- **Container** : runnable instance of an image loaded into memory
- **Image** : file that provides a template for spawning new containers

Docker CLI Essentials

`docker search`	Search Docker Hub for pre-built images
`docker pull`	Pull an image or a repository from a registry
`docker images`	List images
`docker build`	Build an image from a Dockerfile
`docker run`	Run a command in a new container
`docker ps`	List containers
`docker start/stop/restart`	Start/stop/restart a container
`docker exec`	Run a command in a running container
`docker inspect`	Return low-level information on Docker objects
`docker rm`	Remove one or more containers
`docker rmi`	Remove one or more images

Dockerfile Example

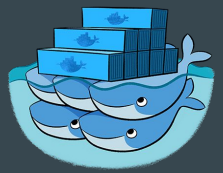
```
# Use Ubuntu as base image
FROM ubuntu:latest
# Update system and install requisite packages
RUN apt-get -y update
RUN apt-get -y install python3 python3-pip
RUN pip3 install --upgrade pip
RUN pip3 install Flask
# Add local files to run the server to our container
ADD hello.py /app/hello.py
WORKDIR /app
# Inform docker we are using port 5000 and run the webserver
EXPOSE 5000
ENV FLASK_APP=hello.py
CMD ["flask", "run", "--host", "0.0.0.0"]
```

Container Orchestration

So how do we actually use containerized apps?

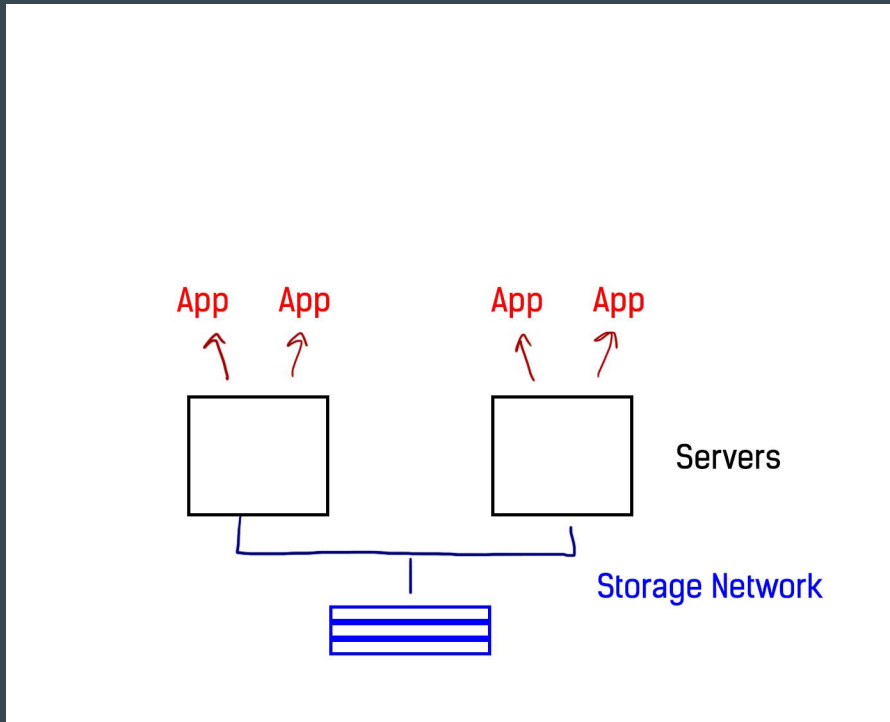
- Container orchestrators help us make sure
  - We have the right amount of containers we want
  - Containers on the proper machines
  - Handle failures by restarting when they die
- *Distributed Systems Magic* ™ to recover from failures
  - Multiple masters, distributed key/value stores, and more!
- Examples: Kubernetes, Mesos + Marathon, Docker Swarm, Kelda





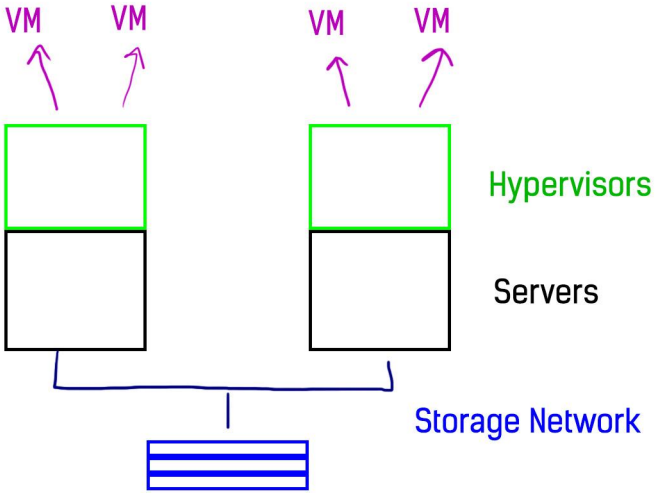

Not-Converged Infrastructure

- Apps run on bare metal
- No isolation
- No differential resource provisioning
- Moving apps is a nightmare
- Compute management is thonk



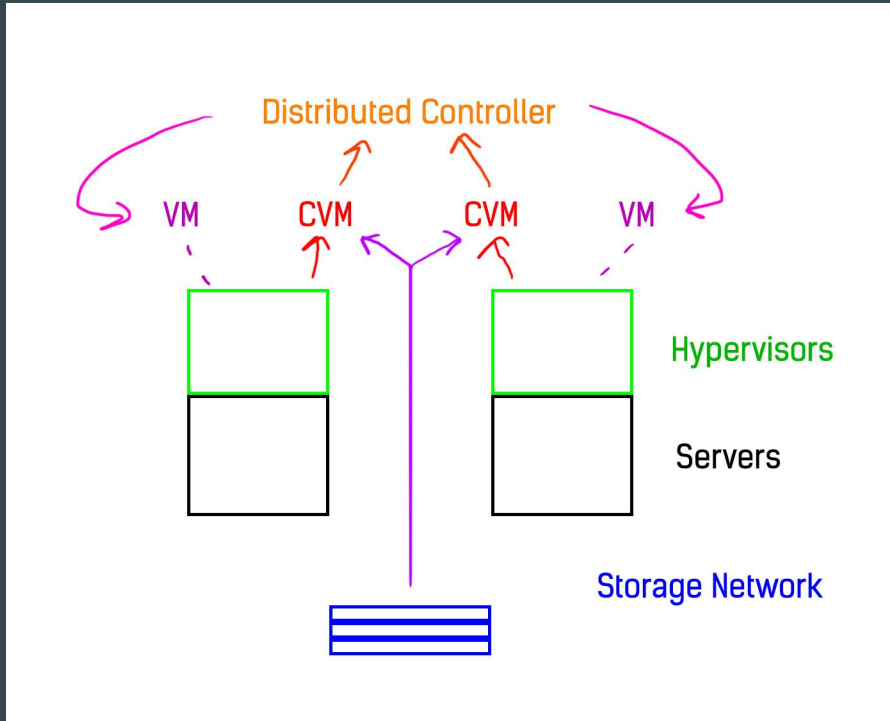
Converged Infrastructure

- Apps run on VMs provisioned by hypervisor
- Isolation
- Differential resource provisioning
- Moving apps is a nightmare
- Compute management is ???



Hyper-converged Infrastructure

- Apps run on VMs provisioned by hypervisor - technically
- Isolation
- Differential resource provisioning
- Software moves apps
- Software compute management
- Scaling is trivial
- Software manages everything



Topics

What's on the menu?

1. Virtual Machines
2. Containers
3. **Configuration**

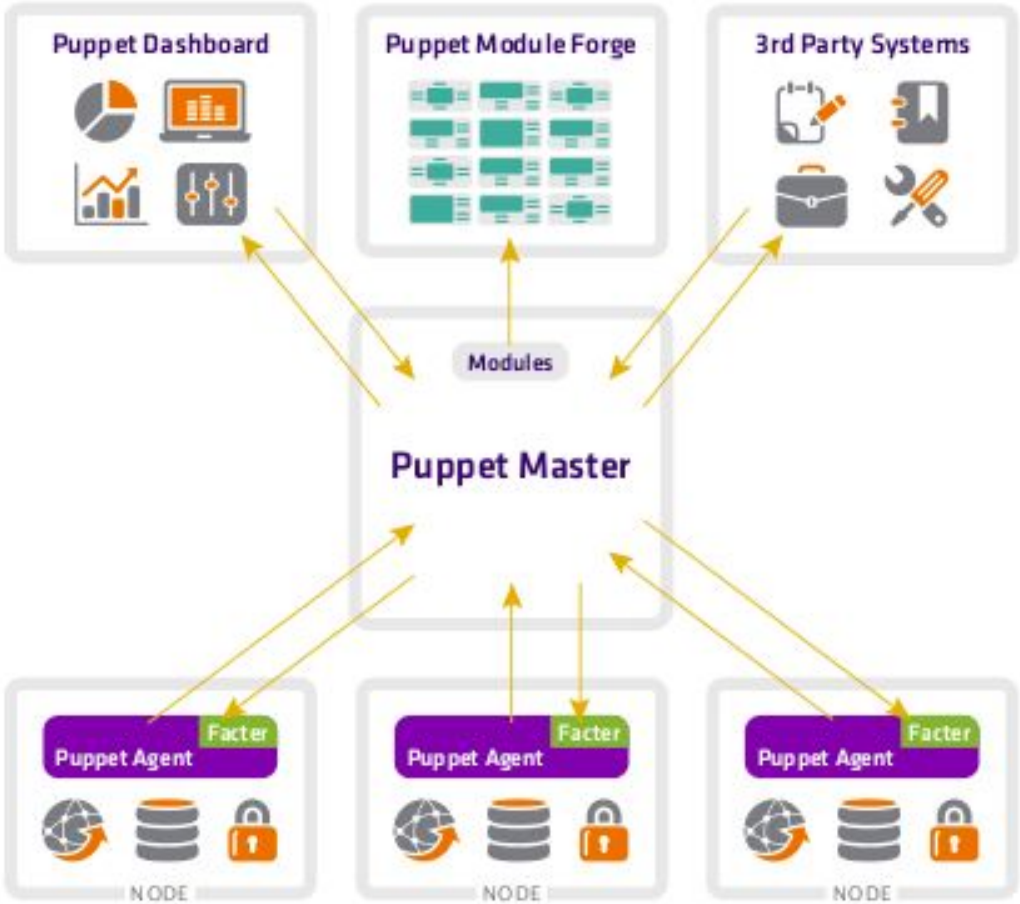
With Scale, New Problems

- How do we provision all XXXX machines so they have everything installed once they boot?
- How do we organize all XXXX machines we have to do work?
- VMs on cloud providers still take 5-10 minutes to provision and boot, how do we react quickly?
- How do we recover from failures automatically?
- What happens when we can’t recover from failures automatically

Automated Configuration Management Tools

- Declarative: Say what you want, not how to do it
  - Application figures out the how
- Can define applications to install, files to include, etc
- Can install different things on different “classes” of machines (desktop vs server)
- Common tools: Puppet, Ansible, Chef

PUPPET PLATFORM

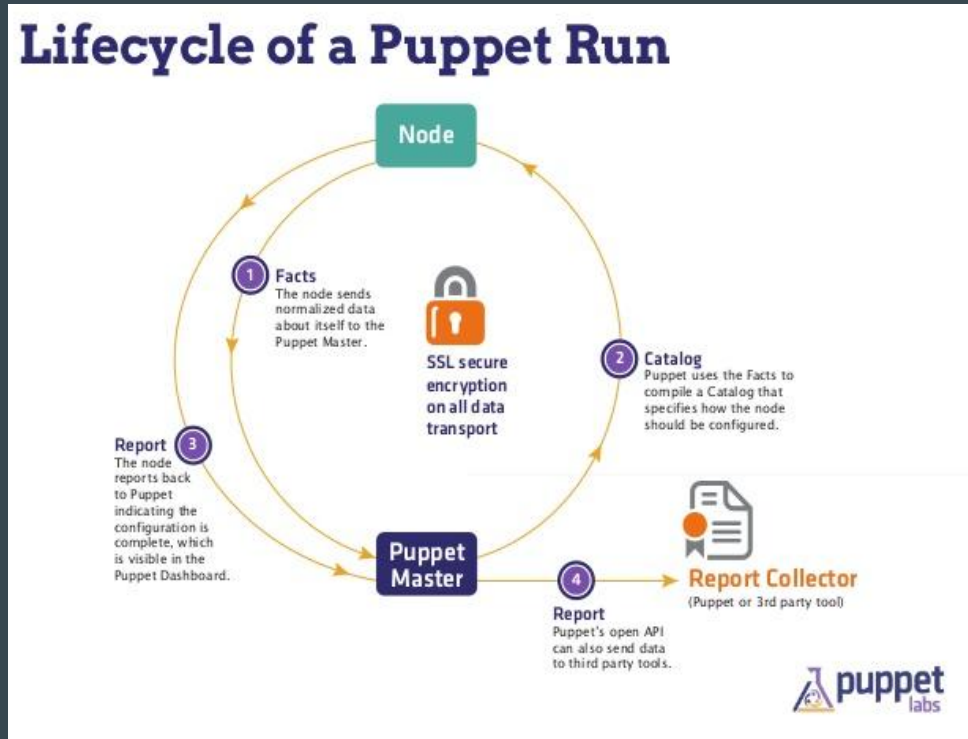


How does Puppet work?

- Agent-master paradigm
  - Puppet agent runs on every machine that is to be configured
  - Master runs on only one machine
  - Master contains all the puppet code we write
  - Code splits into classes, which define different machine configs
    - E.g. mail server
  - Master can even contain code that defines the puppet master config

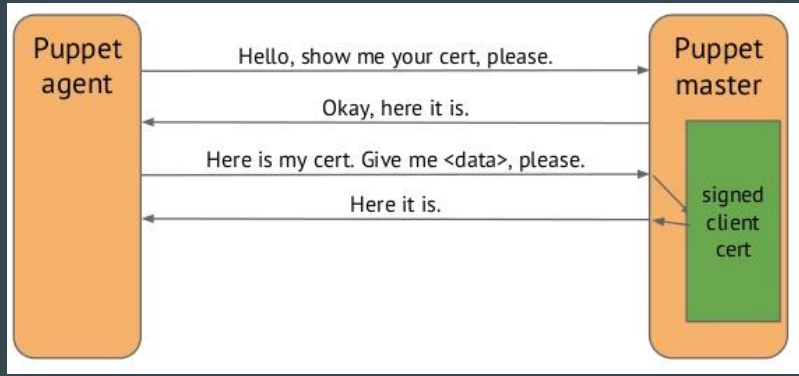
How does Puppet work?

- Applying configurations
  - Master gets node information (facts) from a utility called “facter”
  - Using facts and class type, Puppet compiles manifests into a catalog containing resources and resource dependencies
  - Catalog applied to target, generating a report



How does Puppet work?

- Using SSL Certs for secure communication
  - Based off public key cryptography commonly seen in web applications
- Puppet runs its own CA (certificate authority)
  - On first run an agent must have its cert signed
  - Can have the master autosign, but we want to ensure only legitimate hosts have signed certificates (e.g. DOS attack)



Important Definitions

- Resource - defines a resource we want on the system, such as certain user or file existing, starting a service, installing packages
- Manifest - file with .pp extension, comprised of related puppet resources that achieve a certain task on the target system
- Class - Defined in a manifest, tells Puppet to execute code in that class
  - Normal class declaration: include some_class
  - Resource-like declaration: allow overrides, class {‘some_class’}
- Module - collection of manifests and data (such as facts, files, and templates), and they have a specific directory structure
  - E.g. Mail server module

Example Puppet Resource

- Declaring a user

```
user { 'harry':
  ensure => present,
  uid    => '1000',          # Give harry uid 1000
  shell  => '/bin/zsh',      # Harry's default shell
  home   => '/home/harry'; # Location of harry's home dir
}
```

- How would we give harry group id 2000?

Example Puppet Resource

- Declaring a user

```
user { 'harry':
  ensure => present,
  uid    => '1000',          # Give harry uid 1000
  shell  => '/bin/zsh',      # Harry's default shell
  home   => '/home/harry'; # Location of harry's home dir
}
```

- How would we give harry group id 2000?

```
  gid => ‘2000’,
```

- Other examples

What happens when Puppet fails?

- Failures not graceful, but sane
  - Large report that shows exactly where the error happened
- Puppet runs can fail for many reasons
  - Improper syntax
  - Dependency failure
  - Out of disk space, memory, or other resources
  - Puppet master isn’t running

What happens when Puppet fails?

- Failures not graceful, but sane
  - Large report that shows exactly where the error happened
- Puppet runs can fail for many reasons
  - Improper syntax
  - Dependency failure
  - Out of disk space, memory, or other resources
  - Puppet master isn’t running

Thanks